LESUR Mathieu

AP-4206 - Graphes et algorithme

Construire et optimiser un réseau d'espions partie 2 : arbre de poids minimum

ESIEE Paris - 2022/2023

Sommaire

Introduction

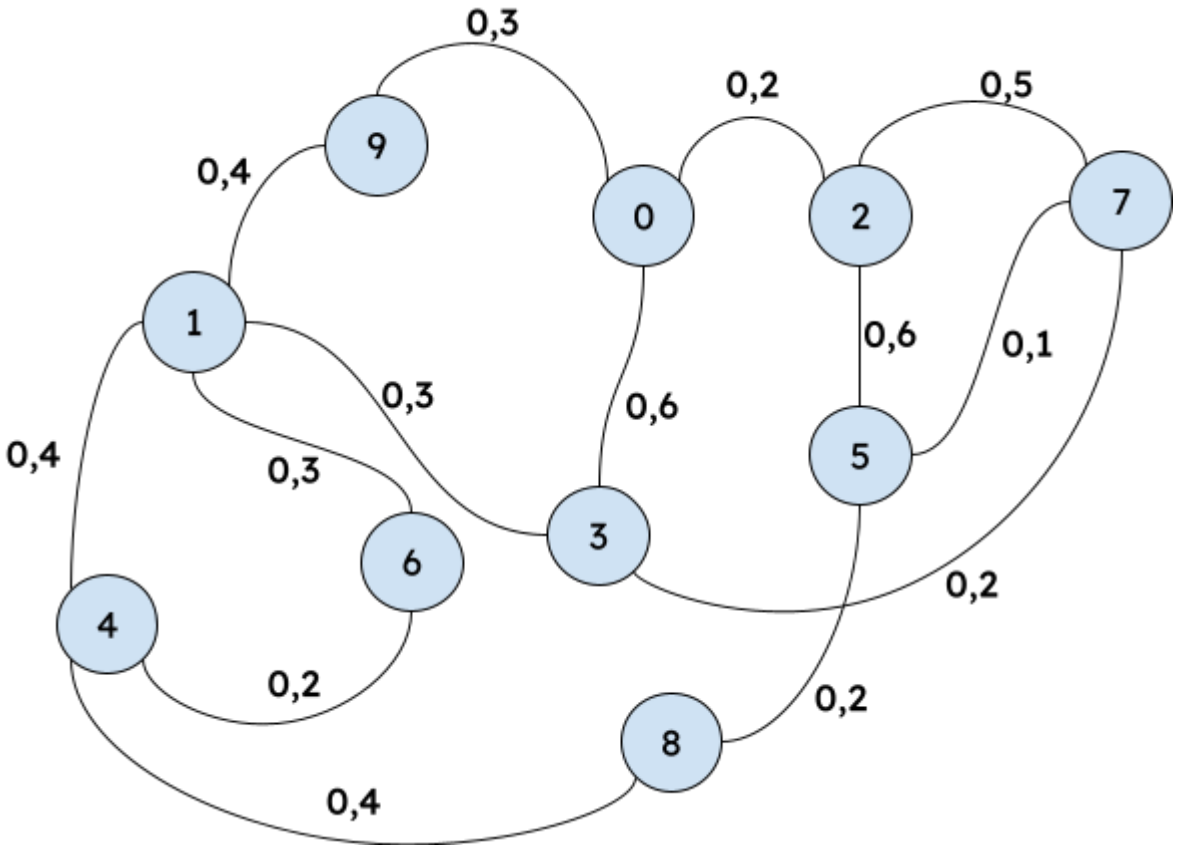
Lien du cours et exercices :

https://perso.esiee.fr/~coustyj/GraphesEtAlgos/2021/html/cours6.html

Lien de mon Github :

https://github.com/MathieuLsr/AP-4206-Graphes-Algorithme/tree/main/Arbres%20poids%20minimum

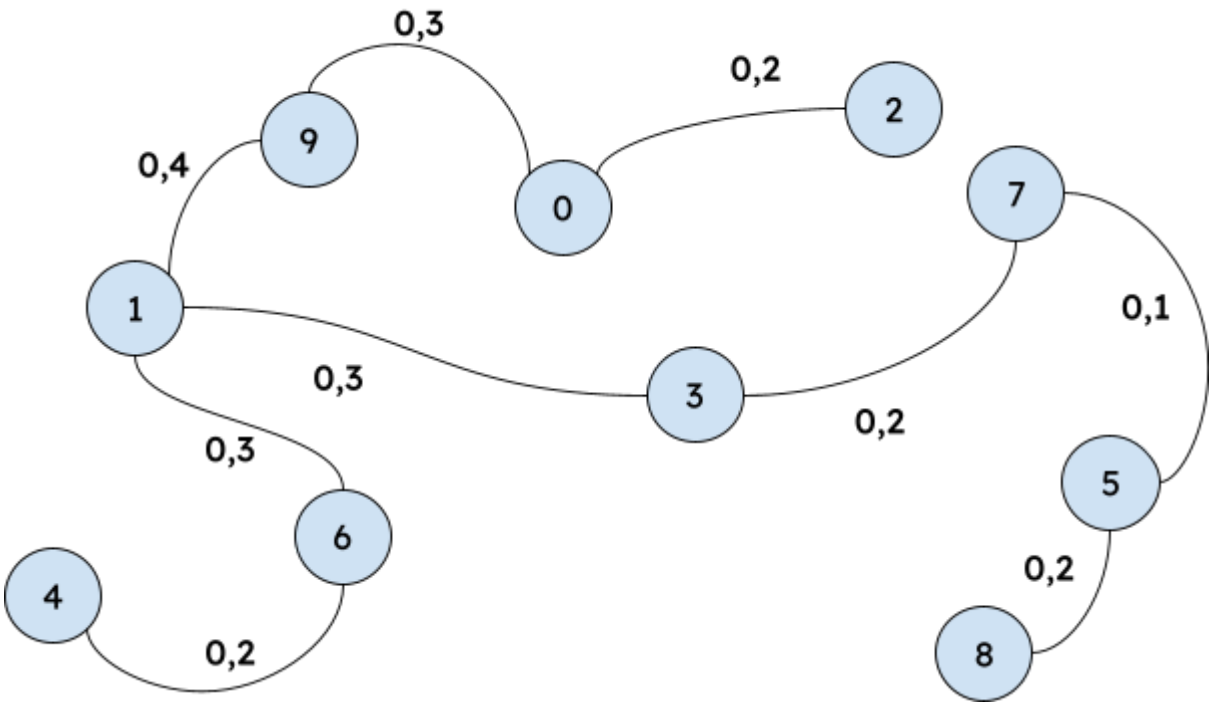
L'exemple jouet :



Exo - Solution optimale pour l'exemple jouet

1. Proposez pour le cas de l'exemple jouet, un schéma de transmission d'un message secret qui minimise la probabilité que le message soit intercepté. Vous n'avez pas besoin ici de justifier que votre proposition est optimale (ce sera l'objet d'une question suivante) ni comment vous avez fait pour arriver à cette solution. On peut donc se contenter d'une solution avec une probabilité d'interception faible (c'est-à-dire pour laquelle on ne trouve pas facilement d'amélioration).

Schéma de transmission d'un message secret qui minimise la probabilité que le message soit interceptée :



2. S'il on voulait vérifier naïvement (avec une méthode force brute) que la solution donnée précédemment est optimale, combien de graphes devrait-on considérer ?

LA méthode de force brute consiste à tester tous les schémas 1 par 1 pour voir lequel est le plus optimal. Or nous savons que si G comporte m arêtes, ce nombre est égal à 2^m donc ici notre graphe comporte 14 arêtes donc 2^14 = 16384 possibilités de graphes.

3. En utilisant le vocabulaire de la théorie des graphes, formulez une solution valide quel que soit l'ensemble des espions et l'ensemble des échanges possibles.

En utilisant le vocabulaire de la théorie des graphes, on peut dire que pour minimiser la probabilité d'interception, il faut choisir le chemin de transmission qui minimise la somme des probabilités d'interception sur tous les arcs du graphe tout en ayant n-1 arcs avec n le nombre de sommets.

4. Justifiez la proposition formulée dans l'exemple précédent. Pour vous aider, vous pouvez vous appuyer sur les rappels suivants :
    - si A et B sont des événements indépendants de probabilités pA et pB alors la probabilité de l'événement (A et B) est égale à pA*pB
    - si A est un événement de probabilité pA, l'événement (non A) est de probabilité 1−pA
    - la fonction logarithme ln est une fonction telle que pour tous réels a et b, nous avons ln(a.b)=ln(a)+ln(b)
    - la fonction logarithme ln est une fonction croissante, ce qui signifie que pour tout couples de réels a et b si a est inférieur à b alors ln(a) est inférieur à ln(b).

$P(Int)$ Probabilité que le message soit intercepté.
$P(InterI)$ Probabilité que le message soit intercepté à la ième arête.

Notre but est d'avoir une probabilité la plus petite possible. Le message intercepté est l'union de probabilité d'interception entre 2 sommets.

Nous voulons utiliser la formule de l'énoncé pour "P(A et B)" donc nous passons par l'évènement contraire de Int :

$$P(\overline{Int}) = 1 - P(\cap\ \overline{InterI}) = 1 - \Pi\, P(\overline{InterI})$$ (via l'énoncé)

Donc :

$$P(Int) \leq \Pi\ P(\overline{InterI})$$

D'après l'énoncé :

$$ln(P(Int)\ ) \leq ln(\Pi\ P(\overline{InterI}))$$

$$ln(\ P(Int)\ )\ \leq \Sigma\, ln(P(\overline{InterI}))$$

$ln(x) \leq 0$ lorsque $0 \leq x \leq 1$ donc :

$$ln(P(\overline{InterI})) \leq P(\overline{InterI})$$

$$ln(P(Int)\ ) \leq \Sigma\ P(\overline{InterI})$$

5. Dans le schéma que vous avez proposé à la question 1, indiquez la probabilité que le message soit intercepté.

Pour avoir la probabilité que le message soit intercepté, nous devons sommer la totalité des poids présents sur le graphe ce qui nous donne 2,2 pour le graphe jouets.

Exo - Calculer des arbres de poids minimum

1. Exécutez à la main l'algorithme de Kruskal sur le graphe de l'exemple "jouet". Pour cela, vous expliciterez l'état des variables à chaque itération de la boucle principale de l'algorithme (conseil : inspirez vous de la présentation du cours ci-dessous).

Dans un premier temps, nous trions nos arêtes selon leur poids en ordre croissant.

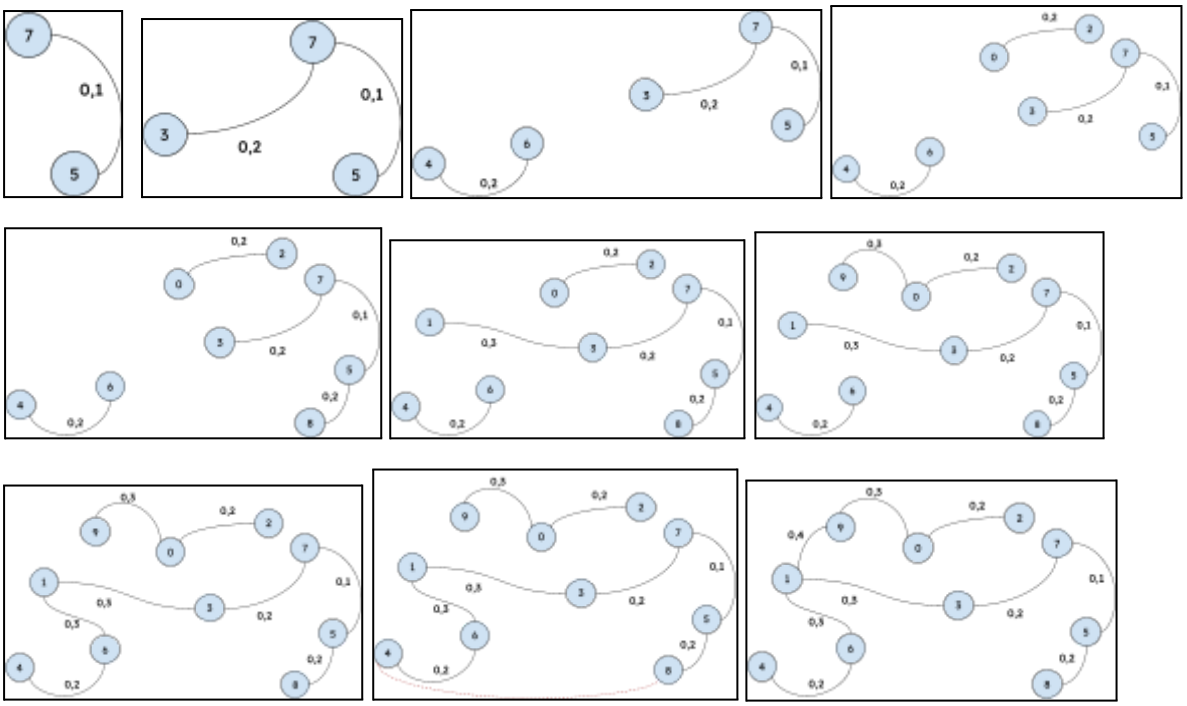
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Sommet A	5	3	4	0	5	1	0	1	4	1	1	2	2	0
Sommet B	7	7	6	2	8	3	9	6	8	9	4	7	5	3
Poids	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.5	0.6	0.6

Ensuite, nous faisons fonctionner notre algorithme de Kruskal :

i	0	1	2	3	4	5	6	7	8	9
Sommets	5 7	3 7	4 6	0 2	5 8	1 3	0 9	1 6	4 8	1 9
Poids	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0 (0.4)	0.4
PoidsArbre	0.1	0.3	0.5	0.7	0.9	1.2	1.5	1.8	1.8	2.2

Pour l'indice i=8, nous ne pouvons pas ajouter l'arête 4-8 au graphe sinon le graphe comporte un cycle donc nous n'ajoutons pas cette arête.

Un schéma a été fait par indice. Nous voyons l'évolution du graphe au fil de la boucle :



2. Indiquez si la proposition que vous aviez faites précédemment (c'est-à-dire dans la section Exo précédente) pour le réseau d'espions était correcte.

Mon graphe était correct car la logique utilisée était la même : enlever les arcs avec le poids les plus élevés.

3. Implémentez l'algorithme de Kruskal.

L'implémentation de l'algorithme de Kruskal est disponible sur mon Github ou alors en annexe :

```
graphe * algoKruskal(graphePondere *Gp){
 graphe *T; /* pour stocker l'arbre de poids minimum */
 int *O;          /* tableau pour ranger les index des arcs par ordre croissant de
poids*/
 int x, y; // index pour des sommets
 booleen *Z; // pour stocker les composantes connexes
 double poidsArbre = 0; // pour calculer le poids de l'arbre de poids min
 int i=0, k=0;

 T = initGraphe(Gp->nsom, Gp->nsom-1);

 O = triAretes(Gp); /* O[i] est l'index de la i-ème arête par ordre croissant de
poids;
   // les extremites de la i-ème aretes sont donc Gp->I[O[i]] et Gp->T[O[i]] */

 /* A completer ici */
 int n = Gp->nsom ;
 while(k < n-1){
   x = Gp->I[O[i]] ;
   y = Gp->T[O[i]] ;
   Z = CC(T, x) ;
   if(Z[y] == FAUX){
     ajouteSuccesseur(T, x, y) ;
     poidsArbre += Gp->poids[O[i]] ;
     k++ ;
   }
   i++ ;
 }

 printf("Le poids de l'arbre de poids minimum vaut : %lg \n", poidsArbre);
 return T;
 }
```

4. Testez votre implémentation de l'algorithme de Kruskal sur espions.graphe et autres.

Voici les résultats obtenus sur les différents fichiers fournis. Je constate de bons résultats et de bonnes performances dû à l'algorithme.

Ce premier lancement du programme avec les débugs nous permet de visualiser si notre programme se comporte correctement :

```
MacBook-Air:Arbres poids minimum mathieulesur$ ./arbrePoidsMin espions.graphe resultats.graphe
espions.graphe lu
Ajoute : 5 7, 0.100000
Ajoute : 3 7, 0.200000
Ajoute : 4 6, 0.200000
Ajoute : 0 2, 0.200000
Ajoute : 5 8, 0.200000
Ajoute : 1 3, 0.300000
Ajoute : 0 9, 0.300000
Ajoute : 1 6, 0.300000
Ajoute : 1 9, 0.400000
Le poids de l'arbre de poids minimum vaut : 2.2
APmin calcule en 0.000198 secondes
```

Ici, nous lançons notre programme sans l'affichage du debug pour ne pas compromettre les temps du programme :

```
MacBook-Air:Arbres poids minimum mathieulesur$ ./arbrePoidsMin espions.graphe resultats.graphe
espions.graphe lu
Le poids de l'arbre de poids minimum vaut : 2.2
APmin calcule en 0.000082 secondes

MacBook-Air:Arbres poids minimum mathieulesur$ ./arbrePoidsMin aleaPondere_1000_8000.graphe
resultats_1000_8000.graphe
aleaPondere_1000_8000.graphe lu
Le poids de l'arbre de poids minimum vaut : 70.36
APmin calcule en 0.249772 secondes

MacBook-Air:Arbres poids minimum mathieulesur$ ./arbrePoidsMin aleaPondere_10000_80000.graphe
resultats_10000_80000.graphe
aleaPondere_10000_80000.graphe lu
Le poids de l'arbre de poids minimum vaut : 702.97
APmin calcule en 45.241141 secondes
```

5. Utilisez vos implémentations de l'algorithme de Kruskal et de l'algorithme d'arborescence vu en partie 1 pour donner la liste des "communications" à établir entre les agents pour faire circuler le message secret depuis l'agent 007 en minimisant la probabilité que le message soit intercepté.
    - Pour chaque communication, vous afficherez l'émetteur et le récepteur du message.
    - Incluez dans votre rapport une trace d'exécution de votre programme.

Dans notre main(), nous voulons utiliser notre méthode arborescence() mais pour cela, nous devons avoir un graphe symétrique. On peut utiliser la méthode fermetureSymEfficace() mise à disposition pour obtenir un graphe symétrique.
Après ça, nous utilisons notre méthode arborescence pour remplir notre tableau des prédécesseurs.
Pour faciliter la programmation, nous définissons la variable START_INDICE préalablement à 7 pour commencer à l'indice 7.

```
debut =  clock();

 T = algoKruskal(Gp);        /* traitement : calcule le symetrique de g */

 fin = clock();

 int pred[T->nsom] ;
 //for(int indice = 0 ; indice < T->nsom ; indice++) pred[indice] = 0 ;
 pred[START_INDICE] = -1 ;

 graphe* T_sym = fermetureSymEfficace(T) ;

 Z = arborescence(T_sym, START_INDICE, pred);

 for(int i=0 ; i < T_sym->nsom ; i++)
     printf("> Le sommet %d reçoit l'information via le sommet %d\n", i, pred[i]) ;

 printf("\n------\n\n") ;

 for(int i=0 ; i < T_sym->nsom ; i++)
   for(int j=0 ; j < T_sym->nsom ; j++){
        if(pred[j]  ==  i)  printf(">> Le sommet %d envoie l'information à %d.\n",
pred[j], j) ;
   }
```

Comme pour le TP précédent, nous mettons en forme le tableau reçu en affichant dans un premier temps par qui chaque sommet reçoit l'information puis dans un second temps qui envoie à qui.

Nous obtenons le résultat suivant :

```
MacBook-Air:Arbres poids minimum mathieulesur$ ./arbrePoidsMin espions.graphe resultat_jouet_2.graphe
espions.graphe lu
Le poids de l'arbre de poids minimum vaut : 2.2
> Le sommet 0 reçoit l'information via le sommet 9
> Le sommet 1 reçoit l'information via le sommet 3
> Le sommet 2 reçoit l'information via le sommet 0
> Le sommet 3 reçoit l'information via le sommet 7
> Le sommet 4 reçoit l'information via le sommet 6
> Le sommet 5 reçoit l'information via le sommet 7
> Le sommet 6 reçoit l'information via le sommet 1
> Le sommet 7 reçoit l'information via le sommet -1
> Le sommet 8 reçoit l'information via le sommet 5
> Le sommet 9 reçoit l'information via le sommet 1

------

>> Le sommet 0 envoie l'information à 2.
>> Le sommet 1 envoie l'information à 6.
>> Le sommet 1 envoie l'information à 9.
>> Le sommet 3 envoie l'information à 1.
>> Le sommet 5 envoie l'information à 8.
>> Le sommet 6 envoie l'information à 4.
>> Le sommet 7 envoie l'information à 3.
>> Le sommet 7 envoie l'information à 5.
>> Le sommet 9 envoie l'information à 0.
APmin calcule en 0.000081 secondes
```

Nous remarquons qu'il n'y a aucune boucle et que chaque sommet reçoit l'information via 1 seul sommet.
En combinant nos 2 algorithmes, nous avons créé dans un premier temps un graphe avec uniquement des chemins minimaux puis dans un deuxième temps, nous avons fait circuler l'information de façon à ce que chaque sommet ne la reçoive qu'une seule fois. Nous avons donc réussi à faire circuler un message secret depuis l'agent i en minimisant la probabilité que le message soit intercepté.